EUROGRAPHICS 2010 / T. Akenine-Möller and M. Zwicker
(Guest Editors)

Volume 29 (2010), Number 2

Real-time Rendering of Heterogeneous Translucent Objects with Arbitrary Shapes

Yajun Wang[†‡§¶] Jiaping Wang[†] Nicolas Holzschuch[‡¶] Kartic Subr[‡¶] Jun-Hai Yong[§] Baining Guo[†§]

[†] Microsoft Research Asia [§] Tsinghua University
[‡] INRIA Grenoble Rhône-Alpes [¶] CNRS and Université de Grenoble, Laboratoire Jean Kuntzman

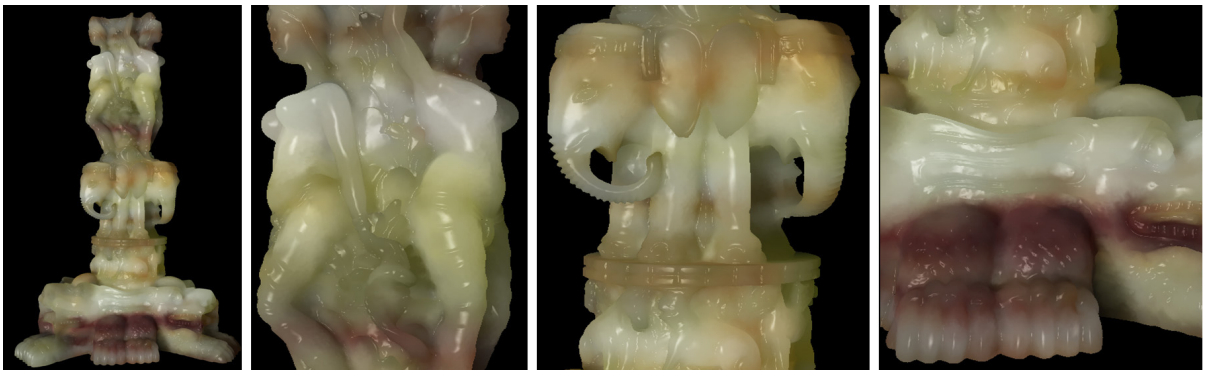

Figure 1: Rendering results at 22 frames per-second of the Stanford Thai Statue (157 K triangles) with our system.

Abstract

*We present a real-time algorithm for rendering translucent objects of arbitrary shapes. We approximate the scattering of light inside the objects using the diffusion equation, which we solve on-the-fly using the GPU. Our algorithm is general enough to handle arbitrary geometry, heterogeneous materials, deformable objects and modifications of lighting, all in real-time. In a pre-processing step, we discretize the object into a regular 4-connected structure (*QuadGraph*). Due to its regular connectivity, this structure is easily packed into a texture and stored on the GPU. At runtime, we use the QuadGraph stored on the GPU to solve the diffusion equation, in real-time, taking into account the varying input conditions: Incoming light, object material and geometry. We handle deformable objects, provided the deformation does not change the topological structure of the objects.*

1. Introduction

Subsurface scattering of light is a complex phenomenon that occurs in many materials such as jade, marble and human skin. It plays an important role in the realism of rendered scenes, but unfortunately is also challenging to simulate. In translucent objects, the outgoing radiance at each point on the surface depends on three factors: Incoming radiance at all points on the surface, the path followed by the light inside the object as well as the optical properties along this path.

Accurately rendering a translucent object can take several hours with off-line physical simulation. To accelerate the rendering process, Jensen *et al.* [JMLH01, JB02] introduced the diffusion approximation: Assuming homogeneous materials and infinite planar boundaries, multiple scattering effects can be simulated efficiently using a dipole approximation. However, the constraints for this diffusion approximation are not fully satisfied by many real-world objects, that have complex shapes and are composed of heterogeneous scattering materials.

The *diffusion equation* [Ish78] fully describes subsurface multiple-scattering effects in translucent objects, including heterogeneous materials. In practice, solving the diffusion requires a discretized model of the interior of the translucent object. Wang *et al.* [WZT*08] used a regular grid, the *polygrid*; this grid had to be built manually, and is not suited for complex geometry (high genus or thin features).

In this paper, we present the first method to solve the diffusion equation, in real-time, on objects of arbitrary shapes with heterogeneous materials. Our method is based on a 4-connected structure, the QuadGraph. In a pre-processing

step, we build the QuadGraph automatically as the connectivity graph of a tetrahedralization of the object. At each node, we store the diffusion coefficients of the material. This QuadGraph is then stored on the GPU in a compact way, exploiting the regular connectivity. At run-time, we use the GPU to solve the diffusion equation at each node of the QuadGraph, in real-time, accounting for dynamic material properties and deformations to the geometry. Our algorithm is robust enough to handle objects with arbitrary genus and arbitrarily thin features (see figures 1 and 10).

We first review the literature on translucent materials and subsurface scattering (section 2). Next, we present the background on the diffusion equation (section 3). In section 4 we describe our datastructure, the QuadGraph, its construction and the linearization of the diffusion equation. Practical implementation details including compact storage of the QuadGraph on the GPU, solving the equation and rendering are explained in section 5. In section 6, we present experimental results and timings. Finally, in section 7, we propose directions for future work.

2. Related Work

Translucent Materials: Subsurface scattering effects can be simulated using Monte-Carlo methods [DEJ*99, PH00, LPT05] or Photon Mapping [JC98]. These methods are physically accurate, and apply to any object or material, but require considerable rendering time for each frame (usually hours). Jensen *et al.* [JMLH01] introduced the diffusion approximation for homogeneous materials with an infinite planar boundary. With this approximation, the rendering time is lowered to minutes. Dachsbacher and Stamminger [DS03] extended this work to achieve real-time rendering, using the same approximation. Several papers have extended this work to multi-layered translucent materials, *e.g.* [DJ05], and especially human skin [GHP*08, DWd*08]. Arbree *et al.* [AWB08] used the lightcuts approach [WFA*05, WABG06] to render objects of arbitrary shape, with homogeneous materials. Several papers have presented algorithms for fast rendering of participating media [MSM*04, ZRL*08, SKLU*09], but they are restricted to regular volumetric shape representation.

Acquisition: A different line of research targets the acquisition of material properties, for translucent objects: Tong *et al.* [TWL*05] for quasi-homogeneous translucent materials, Goesele *et al.* [GLL*04] for acquiring the properties of an entire object, Peers *et al.* [PvBM*06] for capturing the properties of a flat sample. These methods sample the complete light transport operator for the object or for a material sample. Consequently, they store very large data sets that restricts editability of material properties.

Precomputed Radiance Transfer: Recently Precomputed Radiance Transfer [SKS02] has been extended

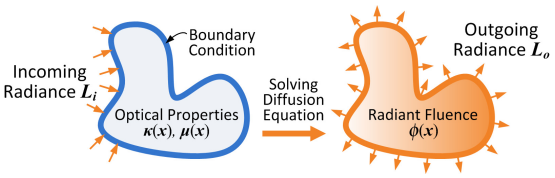

Figure 2: *Rendering with diffusion equation.*

to render translucent objects in real-time, through precomputation of the light transport due to subsurface scattering [HV04, WTL05]. Xu *et al.* [XGL*07] and Wang *et al.* [WCPW*08] precomputed the subsurface scattering of a series of 1D homogeneous basis scattering profiles and linearly combined them on-the-fly with varying weights to achieve runtime alteration of optical properties. These methods require precomputation of the light transport, with fixed material properties and/or geometrical shape of the object. They are not suited for applications that require dynamic editing of the material or the geometry.

Diffusion Equation: Multiple scattering effects can be modeled by the diffusion process [Ish78], described by a PDE– the diffusion equation– that can be numerically solved. The Diffusion Equation was first introduced in computer graphics [Sta95] for rendering participating media. Haber *et al.* [HMBV05] extended this work to objects of arbitrary shapes, using embedded boundary discretization. Wang *et al.* [WZT*08] achieved real-time rendering of translucent objects on the GPU, using a quasi-regular 6-connected structure, the *polygrid*. However, this method is restricted to objects with simple geometry (low genus, no sharp features).

Our algorithm is also based on the diffusion equation, allowing us to handle heterogeneous materials. However, the main difference with previous work is that we discretize translucent objects using tetrahedra instead of cubic grids. This allows us to handle arbitrary geometry, in real-time.

3. The Diffusion Equation

Light entering a translucent object interacts with the material of the object and is scattered several times before leaving the object. If we focus only on this multiply-scattered component, light within the object is related to the light entering the object by a PDE, the diffusion equation [Ish78, Sta95] with incoming radiance being the boundary condition (see figure 2). The solution to the diffusion equation expresses the distribution of light throughout the object and light leaving the object is obtained by computing the outgoing radiance at the surface.

More formally, consider an object Ω with boundary $\partial\Omega$, composed of a highly scattering, non-emissive, heterogeneous material, defined by its absorption coefficient $\mu(x)$ and reduced scattering coefficient $\sigma_s'(x)$ [JMLH01]. The diffusion equation defines the radiant flux $\phi(x)$ within this ob-

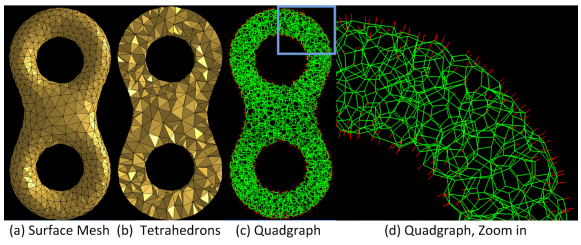

Figure 3: *QuadGraph (c) is constructed from the tetrahedralization (b) of a surface mesh by taking centroid of tetrahedra as nodes and faces as edges. (d) zooms in the blue box region of (c). Green line: a link between inner nodes. Red line: a link between inner node and surface node.*

ject as:

$$\nabla \cdot (\kappa(x)\nabla\phi(x)) - \mu(x)\phi(x) = 0, \quad x \in \Omega \tag{1}$$

where $\kappa(x) = [3\mu(x) + \sigma_s'(x)]^{-1}$.

We use the diffusive source boundary condition as described by Arbree [AWB09] based on the Robin boundary condition [HST*94, SAMD95], which relates the incoming radiance L_i on the surface of the object $\partial\Omega$ to the radiant flux as

$$\phi(x) + 2A\kappa(x)\frac{\partial\phi(x)}{\partial\vec{n}} = \frac{4}{1-F_{dr}}q(x), \quad x \in \partial\Omega \tag{2}$$

where

$$q(x) = \int_{2\pi} L_i(x,\omega_i)(\vec{n}\cdot\omega_i)F_t(\omega_i)d\omega_i\,, \tag{3}$$

$A = (1+F_{dr})/(1-F_{dr})$ and $q(x)$ is the diffused incoming light at the surface point x. F_t and F_{dr} are the diffuse Fresnel transmittance and Fresnel reflectance coefficients, respectively [JMLH01]. Both are functions of the refraction index η.

Once radiant flux $\phi(x)$ inside the object is determined, we compute the outgoing radiance on the boundary $L_o(x_o,\omega_o)$ as its derivative along the normal $\vec{n}$ at point $x_o \in \partial\Omega$ [AWB09]:

$$L_o(x_o,\omega_o) = \frac{F_t(\omega_o)}{4\pi}\left[\left(1+\frac{1}{A}\right)\phi(x) - \frac{4}{1+F_{dr}}q(x)\right] \tag{4}$$

4. Solving the Diffusion Equation on QuadGraph

We represent the boundary condition and solve the diffusion equation in a discrete domain. A regular 4-connected volumetric graph called QuadGraph is proposed to serve as such a domain, which is automatically constructed given the surface geometry of the object. In this section, we describe how to construct the QuadGraph and solve the diffusion equation on it.

4.1. QuadGraph

The QuadGraph is a volumetric graph; nodes are connected to either 1 other node (surface nodes), or to 4 other nodes

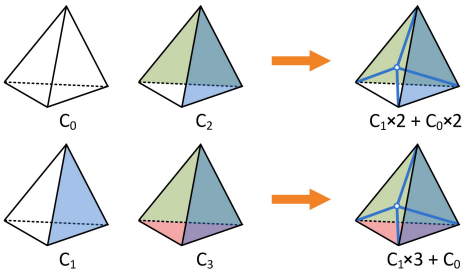

Figure 4: *Four types of tetrahedron (C_{0-3}) with 0-3 outside face(s). Outside faces, rely on the volume boundary, are filled with solid color. C_2/C_3 tetrahedra are split by adding a new vertex and four edges shown in blue. Splitting a C_2 tetrahedron results in two C_0 tetrahedra and two C_1 tetrahedra. Splitting a C_3 tetrahedron results in one C_0 tetrahedra and three C_1 tetrahedra.*

(inner nodes). Surface nodes ($N_{\partial\Omega}$) are positioned at the surface of the object, and they sample the incoming radiance and represent the boundary condition. Inner nodes (N_Ω), distributed inside the volume, sample volumetric optical properties and represent the solution of the diffusion equation, radiant flux. The entire graph (N_{Ω^+}) is the union of surface nodes and inner nodes, $N_\Omega \cup N_{\partial\Omega}$.

As shown in figure 3, to build the QuadGraph, we start with an automatic tetrahedralization of the triangle mesh [ACSYD05, LS07]. After the tetrahedralization, a given tetrahedron can have from 0 to 3 of its faces on the boundary of the object (we assume that there are no isolated tetrahedra).

We split all tetrahedra with 2 or 3 faces on the boundary by adding a new vertex at their centroids as shown in figure 4, so that the resulting tetrahedral mesh only contains tetrahedra with 0 or 1 faces on the boundary.

We then build the QuadGraph as the connectivity graph of this tetrahedral mesh as illustrated in figure 5: each tetrahedron is converted into a graph node, located at its centroid. If two tetrahedra share a given face, then their centroids are connected in the graph. Tetrahedra with one face on the boundary of the object are converted into a set of two connected nodes: an inner node and a surface node. The geometric position of the surface node is determined as the closest intersection between the surface mesh and the ray linking it to the inner node. Figure 3 shows an example of the QuadGraph. Once we have built the initial QuadGraph, we refine

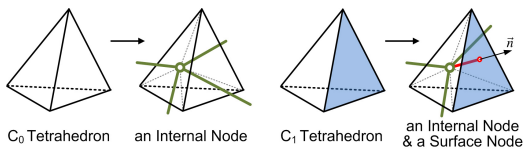

Figure 5: *Construction QuadGraph from tetrahedron mesh. Left: a C_0 tetrahedron is converted to an inner node (Green). Right: a C_1 tetrahedron is converted to an inner node (Green) and a surface node (Red).*

the position of the inner nodes for even distribution by applying reaction-diffusion [Tur91] to the 3D points.

Our method is based on the tetrahedralization method of Alliez *et al.* [ACSYD05], which produces nearly equilateral tetrahedra of regular size. It takes a single parameter, K, which controls how much larger tetrahedra inside the object are, compared to those near the surface.

4.2. Discretized Diffusion Equation

Once we have built the QuadGraph $\{N_\Omega, N_{\partial\Omega}\}$, we discretize the Diffusion Equation (Equations 1 and 2) into linear equations using a Finite Difference Method (FDM), as in [Sta95]:

$$\sum_{j=1}^{4} \frac{1}{d_{ij}^2} \kappa(n_j)\phi(n_j) - (w\kappa(n_i) + \mu(n_i))\,\phi(n_i) = 0 \qquad (5)$$

$$\phi(n_s) + 2A\kappa(n_s)\frac{\phi(n_s) - \phi(n_k)}{d_{sk}} = \frac{4}{1-F_{dr}} q(n_s) \qquad (6)$$

$$n_i \in N_\Omega \quad n_s \in N_{\partial\Omega}$$

The sum in Equation 5 is over the four nodes n_j connected to the node n_i. In Equation 6, n_k is the single inner node connected to the surface node n_s. In equation 5, we use $w = \sum_{j=1}^{4} 1/d_{ij}^2$, and d_{**} denotes the distance between two nodes, or the length of the graph edge connecting them.

We sample optical properties (κ and μ) at the graph nodes. $q(n_s)$ is the diffused incoming light for each surface node. We approximate the coefficients of Laplacian operator with the inverse distance ($\alpha = -2$) [Tau95]. Once we have solved the diffusion equation described in section 4.3, we find the outgoing radiance $L_o(n_s, \omega_o)$ according to the radiant flux of the corresponding surface node $\phi(n_s)$:

$$L_o(n_s, \omega_o) = \frac{F_t(\omega_o)\,(\phi(n_s) - 2q(n_s))}{2\pi(1+F_{dr})}, \quad n_s \in N_{\partial\Omega} \qquad (7)$$

4.3. Solving the linearized equation

We solve the linearized equations (5, 6) on the QuadGraph using the relaxation scheme as in [Sta95, WZT*08]: we start by initializing the value of the radiant flux at each node to $\phi_0(n)$ (e.g. zero), and iterate the following steps until reach a user defined number of iterations.

$$\phi_{t+1}(n_i) = \frac{\sum_{j=1}^{4} \kappa(n_j)\phi_t(n_j)/d_{ij}^2}{w\kappa(n_i) + \mu(n_i)} \qquad n_i \in N_\Omega \qquad (8)$$

$$\phi_{t+1}(n_s) = \frac{2A\kappa(n_s)\phi_t(n_k) + d_{sk}\,\frac{4q(n_s)}{1+F_{dr}}}{2A\kappa(n_s) + d_{sk}} \qquad n_s \in N_{\partial\Omega} \qquad (9)$$

To speed-up the convergence, we use a multi-resolution scheme, similar to [WZT*08]. We design multiple levels h of the QuadGraph, $\{N_\Omega^h, N_{\partial\Omega}^h\}$. Each level is built, independently, as a tetrahedralization of a simpler version of the surface mesh, with the number of surface points decreasing by a factor of four between two hierarchical levels. Note that the finest level uses the exact full QuadGraph, so no approximation is introduced by using the multi-resolution scheme if convergence is fulfilled.

After we have these multi-resolution representations of the object, we start by solving at the coarsest level, $h = 0$, use this solution as the starting point for solving at the next level, and iterate. The coarse level generates a blurred solution (figure 14b) with correct overall distribution and serves as the initialization for the finer level. Specifically, initial radiant flux at the level $h+1$ is sampled from the solution of b nearby nodes on level h, with interpolation weights proportional to the square of the inverse distance [She68]:

$$\phi(n_x^{h+1}) = \sum_{y=1}^{b} w_{xy}\phi\left(n_y^h\right) \qquad (10)$$

$$w_{xy} = \frac{w'_{xy}}{\sum_{y=1}^{b} w'_{xy}} \qquad w'_{xy} = \left\| n_x^{h+1} - n_y^h \right\|^{-2}.$$

We treat surface nodes and inner nodes independently in this hierarchical sampling process. We used $b = 4$ for surface nodes, and $b = 8$ for inner nodes, i.e. each surface node at a finer level is computed by interpolating the values from 4 surface nodes at the coarser level. The interpolation weights between the different levels of the QuadGraph (w_{xy}) are precomputed as part of the pre-processing step. They are independent of the optical properties of the material, and invariant to smooth shape deformation.

5. GPU Implementation and Rendering

Our algorithm solves the diffusion equation inside the translucent object at each frame, without precomputation of the light transport, thus allowing for dynamic illumination, optical properties and shape deformations. We store the QuadGraph in textures on the GPU as a pre-processing step. At each frame, we first sample the incoming illumination using shadow mapping on the surface of the translucent object, then solve the Diffusion Equation on the GPU using it as a boundary condition.

5.1. Storage

We store the QuadGraph itself on the GPU using 2D textures (see figure 6). Each node of the QuadGraph corresponds to

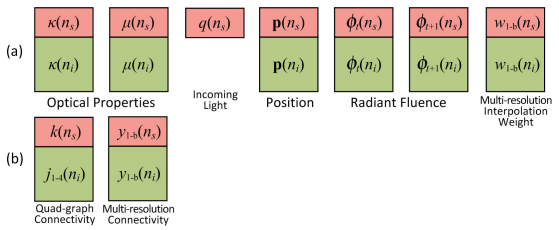

Figure 6: *Textures for encoding parameters. Upper row (a): 16-bits float textures that encode variables. Bottom row (b): 32-bits integer textures that encode constants.*

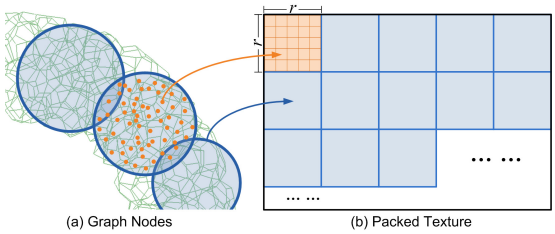

Figure 7: *Texture packing. Topologically nearby nodes (red) are packed in the same texture block (blue).*

a single index (u,v), applying to all the textures. The textures are subdivided along one dimension (v) between surface nodes and inner nodes, so that the node type is easily determined by a simple test.

Connectivity is encoded using two 32-bits RGBA integer textures, encoding for each node the indices of the connected nodes (two channels per index). We also use 2 RGBA textures of 16-bits floating-point numbers to encode the geometric position of each node and its optical properties $(\kappa(n),\mu(n))$. We uses 16-bits floating-point instead of 32-bits in any situation if the precision is sufficient, which reduced the amount of video memory to be accessed.

A separate set of textures encodes the hierarchical levels of the QuadGraph: 32-bits RGBA integer textures to encode the connectivity between the different levels, and 16-bits floating point numbers to encode the weights used for sampling results on the next hierarchical level. The entire GPU memory cost for our algorithm is roughly 20 MB for models with roughly 100 K vertices (see Table 1).

A crucial point for a practical GPU implementation is improving the cache hit rate of graphics memory access on the GPU. We pack together nearby nodes (see figure 7): the texture is subdivided into multiple blocks of $r \times r$ texels, and each block is filled with a set of connected nodes. We use a greedy approach for filling the texture: starting from a seed node, we do a breadth-first traversal on the graph, and assign them to the block until we have reached r^2 nodes. The next node is then used as a new seed node, to fill the next block. When the traversal finds an already assigned node, it does not assign it to another block, but it still visits all the nodes that are connected to it. We iterate this process until all graph nodes have been assigned. In our experiments, we use $r = 32$. Such a chosen is based on the tetrahedralization density of the geometry models used in our experiments.

For packing together inner nodes, we use the connectivity inside the graph. For surface nodes, we use proximity on the surface. In our experiments, this localized storage of the QuadGraph in textures results in a 30 % to 60 % speedup over random storage.

bind texture ϕ_{t+1} to a FBO as the render target
render a rectangle covering the texels containing the nodes to be updated, with fragment shader.
for each fragment:
  fetch indices of neighbouring nodes n_j
  fetch positions and optical properties for n_i, n_j
  compute Laplace weights based on distance between nodes
  compute value for ϕ_{t+1} based on Eq. 9 or 8.
end for

Figure 8: *Pseudocode for a single iteration*

5.2. Solving the Equation

We solve the diffusion equation at each frame, recursively on each hierarchical level of the QuadGraph, starting with the coarsest level ($h = 0$) (see figure 9 for a pseudo-code of our algorithm).

For each level, we start by evaluating the incoming diffuse illumination on the surface, including shadows (using shadow mapping) and multiply with Fresnel transmission coefficient . Incoming light is stored in a 16-bits floating point texture, for each surface node of the QuadGraph.

We then solve the diffusion equation iteratively, using two auxiliary textures to store the radiant flux, before and after the current iteration (see figure 8 for the pseudo-code of a single iteration). After computing an iteration, we test for convergence. If we have converged at this hierarchical level, we move on to the next level, using the results at this level as a starting point, after re-sampling. Otherwise, we swap the two textures and compute the next iteration.

5.3. Displaying Results

We do not display the geometry of the original object; instead, we build a surface Υ that maps directly to the QuadGraph, by triangulating all its surface nodes. Once we have computed the solution of the Diffusion Equation, we use it to compute the outgoing radiance L_o for all surface nodes n_s using Eq. 7, then we render Υ.

6. Experimental Results

We have implemented our algorithm on an Intel Core2Duo 2.13GHz CPU, with 2GB memory and an NVIDIA Geforce 8800GTX GPU with 768MB graphics memory. We used GLSL to implement the algorithm on the GPU.

At each new frame:
for each multi-resolution level h
  compute incoming light $q(n_s)$ on all surface nodes
  initialize $\phi(n)$ by sampling from level $h-1$
  while not converged
    update $\phi(n_s)$ for all surface nodes (Eq. 9)
    update $\phi(n_i)$ for all inner nodes (Eq. 8)
  end while
end for
compute radiance $L_o(n_s,\omega_s)$ for all surface nodes (Eq. 7)
render the object surface using L_o

Figure 9: *Pseudocode for our linear solver.*

Scene	Fig.	QuadGraph				Mem (MB)	Prep (min)	Iter	FPS
		N_s	N_i	K	L				
Thai	1	157k	369k	1	4	35	24	40	21.8
Gargoyle	10a	62k	139k	0.5	3	18	10	50	46.4
Topmod	10b	121k	260k	0.2	3	36	15	40	29.4
Buddha	10c	66k	120k	0.5	3	17	9	30	57.6
Fertility	10d	82k	182k	0.3	3	25	12	50	34.4
Twirl	10e	82k	226k	0.1	3	27	16	60	22.1
Heptoroid	10f	82k	178k	0.2	2	24	11	80	21.6
Lucy	10g	150k	363k	0.2	4	33	20	40	21.0

Table 1: *Statistics of test scenes: N_s and N_i are the number of surface and inner nodes, respectively, of the QuadGraph at the finest level. L is number of multi-resolution levels;* Mem *is the total size of textures over all levels;* Prep *is the preprocessing time including tetrahedralization, QuadGraph construction and texture packing;* Iter *is the number of iterations of the solver at each level.*

All images are rendered at 1024×1024 with 2×2 supersampling for antialiasing. We added surface shading to all results using the Cook-Torrance BRDF model [CT81]. We have tested our algorithm on several complex objects, with sharp geometric features (see figures 1 and 10). Rendering statistics for all our models can be found in Table 1.

6.1. Test Scenes

Figure 1 shows the rendering results of the Stanford Thai model with the colorful agate material. Our system generates the subsurface effects due to the highly heterogeneous material. Complex surface details and small geometry features, such as the trunk in the middle and claw in the bottom, are well preserved and exhibit rich visual effects with different lighting conditions. Please refer to the accompanying video for animated sequences.

Figure 10 shows rendering results with various objects. Objects with high genus (b, c, f), high curvature (b, e, f) and thin features (a, c, e, g) are well handled. We can also handle any kind of material and heterogeneities, such as wax (a, c, e, f), jade (b), and marble (d, g). Our algorithm produces convincing results in all cases, which demonstrates various visual effect on translucent materials, such as color bleeding (a, d, e, g), back-lighting (b, c, d, f) and blurring of the geometry features (a, c, g).

6.2. Computation time

Figure 12 shows the computation time (in ms) for several test scenes, as a function of the number of tetrahedra. For three of our models, the Stanford Bunny, the double-hole donut (figure 3) and the Fertility statue (figure 10d), we have several tetrahedralization: they are plotted as lines. Other models, for which we have a single tetrahedralization, appear as points on the graph. For most of the models, the rendering time appears to vary linearly with respect to the number of tetrahedra. The two outliers are the Twirl (figure 10e) and the Heptoroid (figure 10f). These two models have thinner features, and require more iterations for convergence.

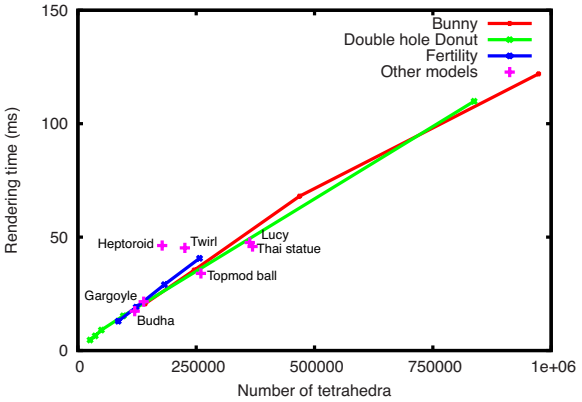

Figure 12: *Computation time, in* ms*, as a function of the number of tetrahedra. Models with several tetrahedralizations are plotted as straight lines (Bunny, Donut, Fertility). Other models appear as points in the graph. For most of the models, the rendering time varies linearly with respect to the number of tetrahedra. The two outliers (Twirl and Heptoroid) have thinner features and require more iterations.*

6.3. Comparison with Ground Truth

To study the effect of the parameter K on the results of our algorithm, we compared resulting images for different values of K against a ground truth reference image that was computed using photon mapping. Figure 11 tabulates these results for four different values of K and shows difference images to the reference and scanline plots. A lower value for K constrains the sizes of surface and internal tetrahedra to vary less. While reducing K increases the number of tetrahedra and hence computation time, it results in rendered images that are closer to the ground truth. However, even the increased quality images render at interactive rates (8 fps). By increasing K, we can further increase the efficiency at the appropriate compromise of fidelity to the ground truth. The plots in the image show our result (blue) across two different scanlines compared against the ground truth (black). For $K = 0.1$, the blue curve is quite close to the reference black curve.

To obtain intuition about the "sweet spot", or the value of K at which we achieve a suitable trade-off between quality and efficiency, we plotted the relative root mean squared (RMS) error of our result (in comparison with the ground truth) against K for multiple models (see figure 13). This plot reveals that even at $K = 1.0$ the RMS error is less than

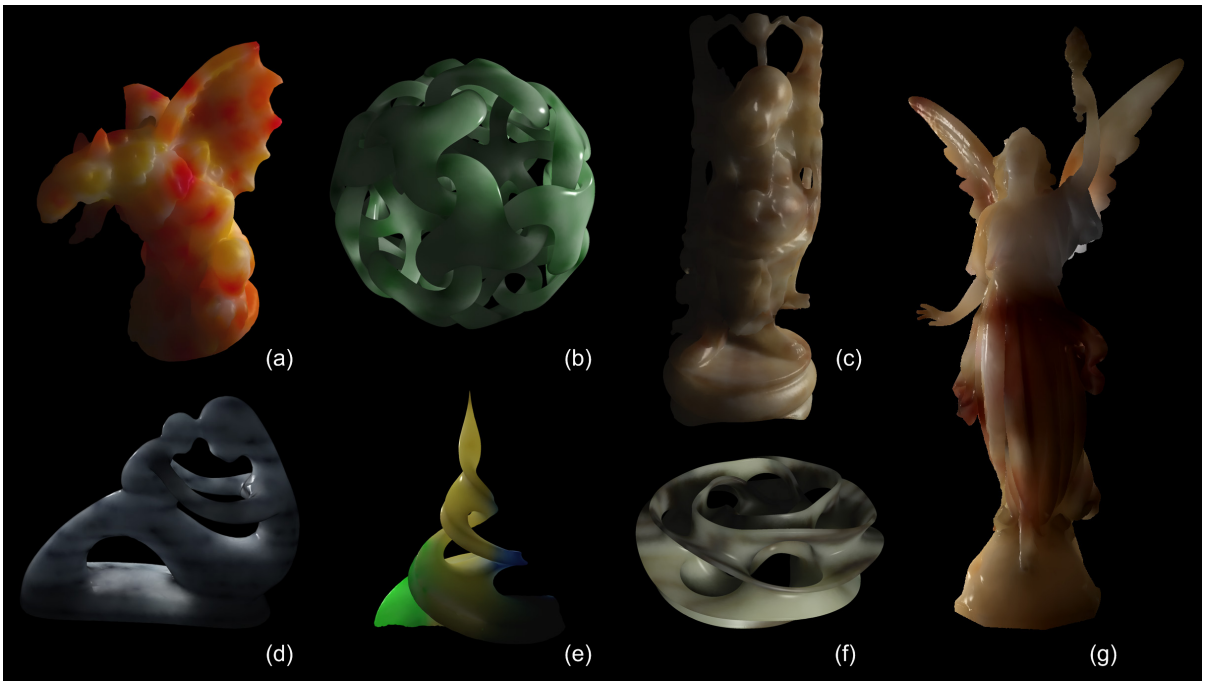

Figure 10: *We have tested our algorithm on several complex objects, including objects with high genus (b, c, f), high curvature (b, e, f) and with sharp geometric features (a, c, e, g).*

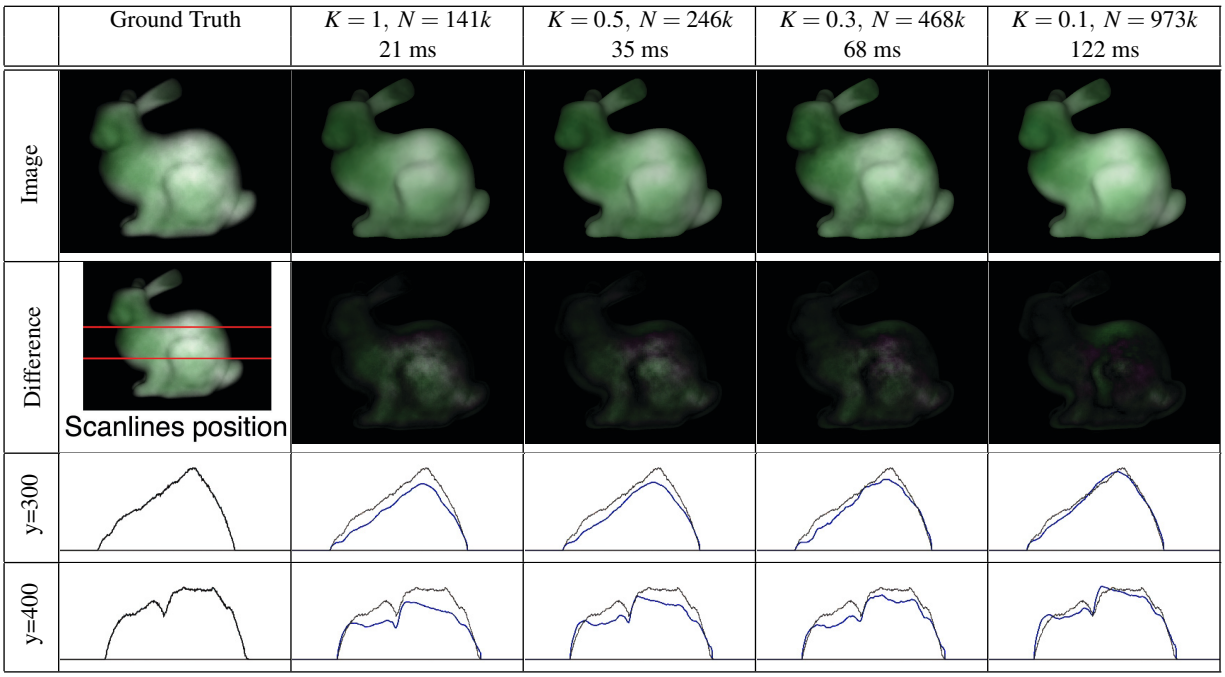

Figure 11: *Comparison between ground truth and our method, for different values of K [ACSYD05]. For this figure, we computed only the translucency effects, with no reflections on the surface, to ensure a fair comparison. First row: the actual pictures computed. Second row: pixel-by-pixel difference with reference image (top left). Third and fourth row: 1D plot of pixel values along a scanline. The positions of the two scanlines appear on the leftmost image of the second row. The values for the reference image are in black, our results are in blue. For $K = 0.1$, our results are quite close to the reference. Note that for all values of K, our algorithm finds the salient features of illumination, resulting in a good visual quality.*

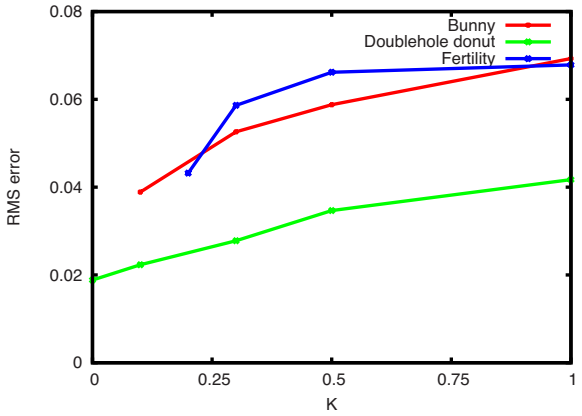

Figure 13: *RMS error of the images computed with our algorithm (compared to a reference image), as a function of K.*

10 %. The plot also shows that we can achieve errors as low as 2 % at slight costs in efficiency.

Finally, we verified the contribution of the multi-resolution scheme to convergence rate by plotting relative errors against execution time, with and without the multi-resolution solver. Figure 14 shows that the use of a multi-resolution solver significantly increases convergence rate.

6.4. Shape Deformation and Material Editing

Our system also supports runtime shape deformation as shown in figure 15 and the accompanying video. Any deformation algorithm that is able to apply a deformation field to the object volume (e.g. [SP86, JSW05, LLCO08]), can be applied to drive the deformation of QuadGraph. Deformation described here is only changing on geometry shape, the material changing due to deformation (e.g. condensed by squeezing) is not considered. Runtime editing of the optical

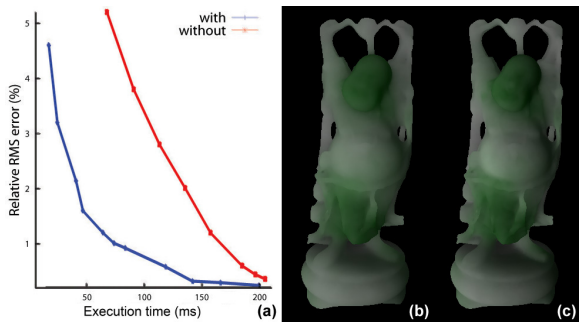

Figure 14: *The plot (a) compares relative error achieved with (blue) and without (red) the multi-resolution solver. (b) the solution of the coarse level. (c) the solution of the finer level with (b) as initialization.*

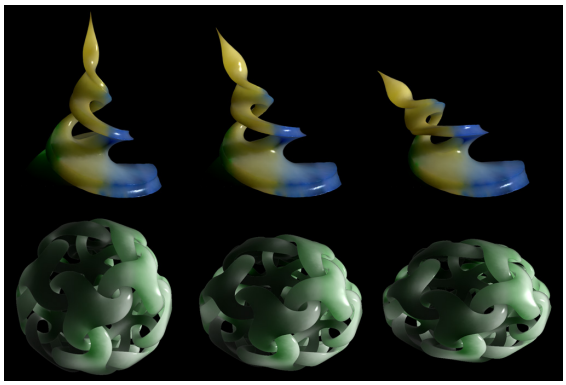

Figure 15: *Our algorithm can handle deforming objects, as long as the deformation does not break the underlying tetrahedralization.*

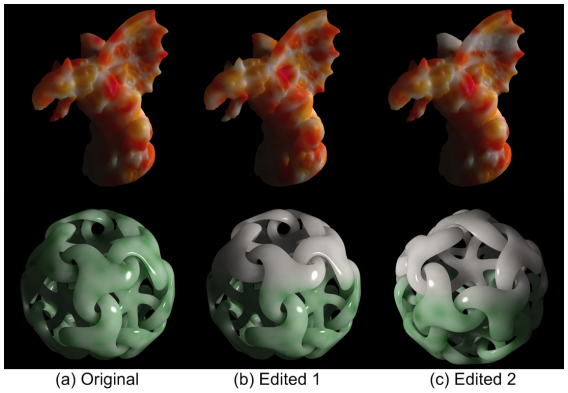

Figure 16: *Our algorithm can handle changing the optical properties at runtime.*

properties is demonstrated in figure 16 (and accompanying video)

We can only handle deformations that maintain the underlying tetrahedralization. Deformations are therefore limited in scope and amplitude: we cannot change the topology of the object.

Although we only show simple material optical editing operations, our system is flexible enough to work with complicated editing operations since the rendering takes the optical property volume as input directly.

6.5. Discussion

Real-time solver: The main advantage of our QuadGraph algorithm is its ability to solve the diffusion equation in real-time. Since there are no precomputations besides the tetrahedralization of the input geometry, our algorithm can handle dynamically changing the incoming lighting, material properties, and even limited deformations of the geometry, without the need for precomputing the light transport.

Quality *vs.* efficiency: The parameter K can be used to trade-off accuracy for efficiency: large values of K result in a smaller number of inside tetrahedra, thus making the algorithm run faster, at the expense of accuracy. Small values of K give results that are more accurate, at the expense of rendering time. But even in its fast, low-quality version, our algorithm identifies the salient lighting features, resulting in a good visual quality, and it maintains RMS error under 10 % (see figure 13).

Sampling: Just as any discretization scheme, our Quad-Graph domain requires adequate sampling for capturing high-frequency effects. That is, the tetrahedralization needs to be dense enough to capture variations in (1) incident illumination at the surface as well as (2) the optical properties of heterogenous materials. The former is less of a problem since multiple scattering results in a decimation of high-frequencies in the incident surface radiance. The latter problem limits the range of editable materials that can be faithfully rendered without re-tetrahedralization.

Arbitrary shape: The dipole approximation [JMLH01] breaks down at points with high curvature, while the polygrid method [WZT*08] is unable to handle thin features or geometries with high genuses. Since our discretization is based on the input geometry, our algorithm is robust to variations in these features.

Discretization method: Our algorithm uses a Finite Difference Method for discretizing the diffusion equation. Other algorithms have used a Finite Element Methods instead [AWB08, AWB09]. The latter is more accurate, but the former maps well to the GPU, allowing us to better take advantage of the computing power of this architecture.

7. Conclusion and Future Work

In this paper, we introduced an algorithm for real-time rendering of translucent objects, of arbitrary shape, with heterogeneous materials. Our algorithm is robust enough to handle objects with arbitrary genus and fine sharp features. Since all illumination computations are conducted in real-time, we can handle deformable objects as well as dynamic changes in the optical properties and lighting conditions. Our algorithm is based on a simple 4-connected graph, the Quad-Graph, built automatically from a tetrahedralization of the triangle mesh defining the surface of the object.

As most existing algorithms for approximating subsurface scattering, our algorithm is restricted to the multiple scattering component of subsurface scattering. While this component is usually the most visible for optically dense materials, recent research [DLR*09, WZHB09] have shown that low-order scattering effects can be quite impressive in some circumstances. In future work, we will investigate computation of low-order scattering effects for increased realism.

Although our algorithm handles deformable objects, deformations are limited, in the sense that the underlying tetrahedralization must remain valid. We cannot change the topology of the object, or break it into pieces. Future work will include handling arbitrary deformations of the object.

Finally, we would like to extend our algorithm to decouple the local scattering and handle it by texture space filtering as in [dLE07], for fine detailed material variation.

Acknowledgements: We are grateful to Adam Arbree, Bruce Walter and Kavita Bala for sharing their latest work on the new diffusion equation formula and the helpful discussions on its accuracy and robustness. We thank Shuang Zhao and Yue Dong for their source code, as reference, for the forward diffusion equation and photo-tracing. We also thank Jean-Claude Paul for the helpful discussions.

The authors from Tsinghua University were supported by the National Science Foundation of China (60625202, 60911130368) and Chinese 863 Program (2007AA040401). This work was funded in part by ANR (ANR-07-BLAN-0331 "HFIBMR"). Yajun Wang is supported by an INRIA internship grant.

Geometry model credits: Gargoyle, fertility and twirl in figure 10(a,d,e) are downloaded from the AIM@SHAPE Shape Repository (http://shapes.aim-at-shape.net/). Thai Statue, Buddha and Lucy in figure 1 and figure 10(c,g) are downloaded from the Stanford 3D Scanning Repository (http://graphics.stanford.edu/data/3Dscanrep/). Heptagonal Toroid in figure 10(f) is generated with Carlo Séquin's sculpture generator (http://www.eecs.berkeley.edu/~sequin /SFF/spec.heptoroid.html). Topmod Ball in figure 10(b) is obtained from David Morris and is originally created by Torolf Sauermann (http://www.evolution-of-genius.de/).

References

[ACSYD05] ALLIEZ P., COHEN-STEINER D., YVINEC M., DESBRUN M.: Variational tetrahedral meshing. *ACM Trans. Graph. 24*, 3 (2005), 617–625.

[AWB08] ARBREE A., WALTER B., BALA K.: Single-pass scalable subsurface rendering with lightcuts. *Computer Graphics Forum (Eurographics'08) 27*, 2 (2008), 507–516.

[AWB09] ARBREE A., WALTER B., BALA K.: Diffusion formulation for heterogeneous subsurface scattering. *Cornell Computer Science Technical Report, Cornell University (http://hdl.handle.net/1813/14199)* (Dec 2009).

[CT81] COOK R. L., TORRANCE K. E.: A reflectance model for computer graphics. *Computer Graphics (SIGGRAPH '81) 15*, 3 (1981), 307–316.

[DEJ*99] DORSEY J., EDELMAN A., JENSEN H. W., LEGAKIS J., PEDERSEN H. K.: Modeling and rendering of weathered stone. In *Proc. ACM SIGGRAPH* (1999), pp. 225–234.

[DJ05] DONNER C., JENSEN H. W.: Light diffusion in multi-layered translucent materials. *ACM Trans. Graph. 24*, 3 (2005), 1032–1039.

[dLE07] D'EON E., LUEBKE D., ENDERTON E.: Efficient Rendering of Human Skin. In *Rendering Techniques (Eurographics Symposium on Rendering)* (June 2007), pp. 147–157.

[DLR*09] DONNER C., LAWRENCE J., RAMAMOORTHI R., HACHISUKA T., JENSEN H. W., NAYAR S.: An empirical bssrdf model. *ACM Trans. Graph. 28*, 3 (2009), 1–10.

[DS03] DACHSBACHER C., STAMMINGER M.: Translucent shadow maps. In *Rendering Techniques (Eurographics Workshop on Rendering)* (2003), pp. 197–201.

[DWd*08] DONNER C., WEYRICH T., D'EON E., RAMAMOORTHI R., RUSINKIEWICZ S.: A layered, heterogeneous reflectance model for acquiring and rendering human skin. *ACM Trans. Graph. 27*, 5 (2008), 1–12.

[GHP*08] GHOSH A., HAWKINS T., PEERS P., FREDERIKSEN S., DEBEVEC P.: Practical modeling and acquisition of layered facial reflectance. *ACM Trans. Graph. 27*, 5 (2008), 1–10.

[GLL*04] GOESELE M., LENSCH H. P. A., LANG J., FUCHS C., SEIDEL H.-P.: DISCO: acquisition of translucent objects. *ACM Trans. Graph. 23*, 3 (2004), 835–844.

[HMBV05] HABER T., MERTENS T., BEKAERT P., VAN REETH F.: A computational approach to simulate light diffusion in arbitrarily shaped objects. In *Proc. Graphics Interface* (2005), pp. 79–85.

[HST*94] HASKELL R. C., SVAASAND L. O., TSAY T.-T., FENG T.-C., MCADAMS M. S., TROMBERG B. J.: Boundary conditions for the diffusion equation in radiative transfer. *J. Opt. Soc. Am. A 11*, 10 (1994), 2727–2741.

[HV04] HAO X., VARSHNEY A.: Real-time rendering of translucent meshes. *ACM Trans. Graph. 23*, 2 (2004), 120–142.

[Ish78] ISHIMARU A.: *Wave Propagation and Scattering in Random Media*. Academic Press, 1978.

[JB02] JENSEN H. W., BUHLER J.: A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph. 21*, 3 (2002), 576–581.

[JC98] JENSEN H. W., CHRISTENSEN P.: Efficient simulation of light transport in scenes with participating media using photon maps. In *Proc. ACM SIGGRAPH* (1998), pp. 311–320.

[JMLH01] JENSEN H. W., MARSCHNER S. R., LEVOY M., HANRAHAN P.: A practical model for subsurface light transport. In *Proc. ACM SIGGRAPH* (2001), pp. 511–518.

[JSW05] JU T., SCHAEFER S., WARREN J.: Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3 (2005), 561–566.

[LLCO08] LIPMAN Y., LEVIN D., COHEN-OR D.: Green coordinates. *ACM Trans. Graph. 27*, 3 (2008), 1–10.

[LPT05] LI H., PELLACINI F., TORRANCE K. E.: A hybrid monte carlo method for accurate and efficient subsurface scattering. In *Rendering Techniques (Eurographics Symposium on Rendering)* (June 2005), pp. 283–290.

[LS07] LABELLE F., SHEWCHUK J. R.: Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph. 26*, 3 (2007), 57.

[MSM*04] MAX N. L., SCHUSSMAN G., MIYAZAKI R., IWASAKI K., NISHITA T.: Diffusion and multiple anisotropic scattering for global illumination in clouds. *Journal of WSCG 12*, 1-3 (February 2004).

[PH00] PHARR M., HANRAHAN P. M.: Monte Carlo evaluation of non-linear scattering equations for subsurface reflection. In *Proc. ACM SIGGRAPH* (2000), pp. 275–286.

[PvBM*06] PEERS P., VOM BERGE K., MATUSIK W., RAMAMOORTHI R., LAWRENCE J., RUSINKIEWICZ S., DUTRÉ P.: A compact factored representation of heterogeneous subsurface scattering. *ACM Trans. Graph. 25*, 3 (2006), 746–753.

[SAMD95] SCHWEIGER M., ARRIDGE S., M. H., D.T. D.: The finite element method for the propagation of light in scattering media: Boundary and source conditions. *Medical Physics 22*, 11 (1995), 1779–1792.

[She68] SHEPARD D.: A two-dimensional interpolation function for irregularly-spaced data. In *ACM National Conference* (New York, NY, USA, 1968), ACM, pp. 517–524.

[SKLU*09] SZIRMAY-KALOS L., LIKTOR G., UMENHOFFER T., TÓTH B., KUMAR S., LUPTON G.: Parallel solution to the radiative transport. *Parallel Graphics and Visualization Symposium 2009* (2009).

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph. 21*, 3 (2002), 527–536.

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 151–160.

[Sta95] STAM J.: Multiple scattering as a diffusion process. In *Eurographics Workshop on Rendering* (June 1995), pp. 41–50.

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *Proc. ACM SIGGRAPH* (1995), pp. 351–358.

[Tur91] TURK G.: Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics 25*, 4 (1991), 289–298.

[TWL*05] TONG X., WANG J., LIN S., GUO B., SHUM H.-Y.: Modeling and rendering of quasi-homogeneous materials. *ACM Trans. Graph. 24*, 3 (2005), 1054–1061.

[WABG06] WALTER B., ARBREE A., BALA K., GREENBERG D. P.: Multidimensional lightcuts. *ACM Trans. Graph. 25*, 3 (2006), 1081–1088.

[WCPW*08] WANG R., CHESLACK-POSTAVA E., WANG R., LUEBKE D., CHEN Q., HUA W., PENG Q., BAO H.: Real-time editing and relighting of homogeneous translucent materials. *The Visual Computer Journal (Proceedings of Computer Graphics International 2008) 24*, 7-9 (2008), 565–575.

[WFA*05] WALTER B., FERNANDEZ S., ARBREE A., BALA K., DONIKIAN M., GREENBERG D. P.: Lightcuts: a scalable approach to illumination. *ACM Trans. Graph. 24*, 3 (2005).

[WTL05] WANG R., TRAN J., LUEBKE D.: All-frequency interactive relighting of translucent objects with single and multiple scattering. *ACM Trans. Graph. 24*, 3 (2005), 1202–1207.

[WZHB09] WALTER B., ZHAO S., HOLZSCHUCH N., BALA K.: Single scattering in refractive media with triangle mesh boundaries. *ACM Trans. Graph. 28*, 3 (2009).

[WZT*08] WANG J., ZHAO S., TONG X., LIN S., LIN Z., DONG Y., GUO B., SHUM H.-Y.: Modeling and rendering of heterogeneous translucent materials using the diffusion equation. *ACM Trans. Graph. 27*, 9 (2008).

[XGL*07] XU K., GAO Y., LI Y., JU T., HU S.-M.: Real-time homogenous translucent material editing. *Computer Graphics Forum 26*, 3 (2007), 545–552.

[ZRL*08] ZHOU K., REN Z., LIN S., BAO H., GUO B., SHUM H.-Y.: Real-time smoke rendering using compensated ray marching. *ACM Trans. Graph. 27*, 3 (2008), 36.